DETERMINING STRUCTURAL SIMILARITY IN SEMI-STRUCTURED DOCUMENTS

Field of the invention

The present invention relates to determining structural similarity in semi-structured documents.

Background

Several methods exist that model documents as labeled trees. These methods are based on the fact that any semi-structured document that uses a markup language can be represented as a tree such as a Document Object Model (DOM) tree. The labels of the nodes correspond to the tags in the markup language. These methods define the structural dissimilarity between a pair of documents as the edit distance between the corresponding labeled trees. This is the tree model for the representation of the structural information.

The basic idea behind all tree edit distance algorithms is to find the cheapest sequence of edit operations that will transform one tree into another. Some of these methods model documents as ordered labeled trees, while others model them as unordered labeled trees. In general, finding the edit distance between unordered labeled trees is computationally more complex than finding the edit distance between ordered labeled trees. A key differentiator among the various tree distance algorithms is the set of edit operations allowed.

Some work in this area used insertion and deletion of leaf nodes and relabelling of a node anywhere in the tree. Several other approaches with different sets of edit operations are proposed. These tree edit distance measures have been modified to address issues such as repetitive and optional fields.

For instance, *Nierman et al* [Andrew Nierman, H. V. Jagadish, "Evaluating Structural Similarity in XML Documents", *Proceedings of the Fifth International Workshop on the Web and Databases* (*WebDB 2002*), June 2002] propose a dynamic programming

algorithm that computes the distance between any pair of documents taking into account Extensible Markup Language (XML) issues such as optional and repeated sub-elements. *Andrews et al* further give a method to cluster documents based on this distance measure. The algorithm to compute the tree edit distance for a pair of documents is of quadratic complexity in the combined size of the two documents.

Cruz et al [Isabel F. Cruz and Slava Borisov and Michael A. Marks and Timothy R. Webb, "Measuring Structural Similarity Among Web Documents: Preliminary Results", *Lecture Notes in Computer Science*, volume 1375, page 513, 1998] propose an alternative approach to modeling structure based on tag frequency measures. This approach can be viewed as the node model for the representation of the structural information, since this approach only uses information about the tags of the various nodes in the corresponding tree model.

The method of *Isabel et al* relies on the assumption that tag frequencies reflect some inherent characteristics of Web documents and correlate with its structure. While the node model is very simple, the model does not take into account the order in which tags appear. Therefore, if the tags of all nodes are rearranged, the representation does not change. Thus, the model is adequate only when the templates are drastically different from each other, that is, they have very few tags in common. This is rarely the case in practice.

In view of the above comments, a need clearly exists for an improved manner of comparing documents for determining the structural similarity of the documents.

Summary

Techniques are presented herein for measuring the similarity between two pages based on their structural syntax. Structurally similar pages may differ in their textual and numeric contents. Documents, as well as document collections, are represented as vectors of feature values. These features are based on the words and phrases occurring in the document collection. Therefore, this representation of a document describes text (or possibly semantic) content of documents, and the similarity values describe a type of text or content similarity between the documents.

Several techniques exist to measure similarity between two numeric vectors. Such techniques are used to measure the similarity between two documents, and between a document and a document collection.

For measuring the structural similarity between documents, documents are represented based on their structure. The structure arises from the various elements on the document and the nature of their nesting. After representing documents based on their structure in vector form, an existing method of measuring similarity between vectors is used to obtain the measure of structural similarity between two given documents.

Description of drawings

Fig. 1 is a schematic representation of an XML document and a corresponding labeled tree.

Fig. 2 is a schematic representation of three respective Document Object Model (DOM) trees that are represented for purpose of comparison.

Fig. 3 is a schematic representation of an example DOM tree labeled with positional indices, and is represented for the purpose of discussion.

Fig. 4 is a flowchart of steps in a procedure for comparing a pair of documents.

Fig. 5 is a schematic representation of a computer system suitable for performing the techniques described with reference to **Figs. 1** to **4**.

Detailed description

Two documents are typically assessed to be structurally similar if they have a similar "look and feel", or layout. By way of example, structurally similar pages might be generated in the following ways:

- Pages generated by providing values to a predetermined master shell page.
- Pages dynamically generated on servers using server code
- Pages generated in accordance of a template.

The techniques now described define a procedure for conducting a comparison of documents to reach a quantitative determination of the degree of structural similarity between documents.

Document Object Model

A document is modelled as a labeled tree in the Document Object Model (DOM). In the labeled tree model of a document, each node in the tree corresponds to an element of the markup language in the document. The tag name of an element acts as a label for the node. The inclusion of a tag inside the scope of another tag is captured by a "parent-child" relationship in the labeled tree. Text nodes are excluded from the labeled tree, as text nodes are immaterial to the structural properties of the document. The tree representation of a document is sometimes known as the DOM (Document Object Model) tree.

Fig. 1 depicts an XML document **110** and its corresponding DOM tree **120**. Given a semi-structured document, one can generate its DOM tree manually, or using any suitable parser programme that analyses XML or related documents. Examples of suitable parsers are Xerces, XML4J, and Jtidy, though any suitable parser can of course be used. These and other suitable packages are generally available via the Web. The Jtidy package not only corrects common HTML errors, but also "*XMLizes*" the document and provides the corresponding DOM tree for the document.

Fig. 1 depicts an XML document **110** and its corresponding DOM tree **120**. All other tags appearing in the document are contained within the scope of tag **<a>** (that is, the document tags appear between **<a>** and ****) in the XML document **110,** and therefore the root node in the corresponding DOM tree is labeled as **a** in **120**. Tags **<c>** and **<d>** appear within the scope of tag **<b>** as directly nested tags and therefore nodes 3 and 4, with the labels **c** and **d** respectively, are the children of node 2 in the tree **120**. Two **<e>**

tags and an <f> tag are directly nested inside the <d> tag in **110** and are therefore the children of node 4 are appropriately labeled in **120**.

Bag of Tree Paths Model

In the *bag of tree paths* model, a document is represented by a set of sequences of labels that occur in the paths from the root node to the leaf nodes of the corresponding tree representation (in this case, a DOM tree). The path from the root node to any leaf node contains the root node, the leaf node, and all the intermediate nodes required to reach the leaf node in sequence. Each such path contributes a sequence of labels to the model.

As an example, leaf node 3 of the DOM tree **120** given in **Fig. 1** contributes the sequence **a/b/c**. Similarly, node 5 contributes the sequence **a/b/d/e**. The same sequence of node labels can occur in two or more distinct paths. For example, node 5 and 6 in **Fig. 1** contributes the same sequence of node labels. A sequence of node labels is referred to as a *path*.

Let $D = \{d_1, d_2, \ldots, d_n\}$ be the collection of n trees corresponding to n documents. Let m_i be the number of leaf nodes present in tree d_i. There are thus m_i paths in tree d_i. A dictionary of distinct paths $Dict_{paths}$ can be constructed by collating such paths from all trees d_i ,$1 \leq i \leq n$. Not all paths, however, are equally important for describing the structure of documents in the *bag of tree paths* model. Thus, feature selection techniques are used to remove non-informative paths, to simplify the model. Paths that occur in very few documents, and paths that occur in almost all documents, are desirably eliminated from the dictionary. In general, however, any feature selection method that is deemed suitable can be used.

Let the dictionary of paths after feature selection be $Dict_{paths} = \{p_1, p_2, \ldots, p_N\}$. Let $f_j(p_i)$ denote the frequency of occurrence of path p_i in document d_j, and let $f_{max} = \max_{ij} f_j(p_i)$. Now a document d_j can be represented as a N-dimensional vector $[d_{j1}, d_{j2}, \ldots, d_{jN}]$, where $d_{jk} = f_j(p_k)/f_{max}$, $1 \leq k \leq N$.

The *bag of tree paths* model for a document captures only some of the structural relationships present in the tree. More precisely, the *bag of tree paths* model incorporates all the parent/child relationships, but ignores sibling relationships present in the tree structure.

The similarity between a pair of documents (***d_j***, ***d_l***) is defined as expressed in **Equation [1]** below.

$$sim(d_i, d_l) = \frac{\sum_{k=1}^{N} \min(d_{ik}, d_{lk})}{\sum_{k=1}^{N} \max(d_{ik}, d_{lk})} \qquad \textbf{[1]}$$

The numerator of the right hand side of **Equation [1]** is the sum (over all paths *k in the dictionary of paths)* of the minimum of the two frequencies of occurrence of a path *k* in the two documents ***d_i*** and ***d_l***. This is a measure of how much the two documents have in common in terms of the various paths that appear in the dictionary. The denominator of **Equation [1]** is the sum of the maximum of the frequencies of occurrence over all paths *k*, and serves as a normalizations factor.

The frequencies $f_j(p_i)$ can be ignored, and the occurrence or non-occurrence of the path used. In other words, ***d_j*** is a binary vector. An important aspect of the *bag of tree paths* model is that the model can take into account markup language issues, such as repetition of elements in the similarity measure. Ideally, similarity value between a pair of documents should be high if the documents differ only in the number of times a particular markup language subelement occurs under an element.

Fig. 2 schematically represents three DOM trees **210**, **220**, **230**. A meaningful similarity measure should yield a higher value of similarity for the pair of trees **210** and **230** than for the pair of trees **210** and **220**. All the paths that appear in tree **210** also appear in tree **230**. These trees only differ in the frequency of the path **a/b/e**. By contrast, the tree **210** has two paths that do not appear in tree **220** at all. The proposed similarity measure

determines that two documents that differ only in the frequency of the paths to be more similar than two documents that differ in the occurrence of paths.

The parent-child relationship is sufficient for measuring the structural similarity of documents, if the documents contain many levels of nesting (that is, the depth of the corresponding trees is relatively high). If, however, the documents do not contain many levels of nesting (that is, the corresponding trees are shallow), then the documents will more probably contain the same paths, even if the overall structure differs substantially. For this reason, a preferred model, referred to as the *bag of XPaths* model, is described.

Bag of XPaths Model

The *bag of XPaths* model is described, which captures some sibling information in addition to all the parent/child relationships captured in the *bag of tree paths* model described above.

In the *bag of Xpaths* model, a representation of a document as a labeled tree includes a *positional index* along with the label for each node. The *positional index* of a node *n* advises that the number of previous sibling nodes with the same label as that of node *n*.

Fig. 3 shows an example of a labeled tree 310. Each node in the tree contains a label as well as a *positional index*. The term *index* is used as an abbreviation for *positional index*. An XPath is defined in some contexts to be a path expression that locates nodes in a DOM tree. The term is used herein, however, to mean any mechanism to describe a path in a tree representation of a document that includes, with the label of a node, the information about the number of siblings of the nodes that have the same label as the node. Consequently, an XPath, as used herein, is essentially a form of path representation, in which particular path-specific information is included, in varying degrees of specificity.

The term XPath is defined herein as a sequence of *terms* separated by the character '/'. The syntax for each term is as follows: *term=nodetest[predicate]*. In the existing technical literature, however, the term XPath is used more generically to denote path

expressions that locate nodes in a DOM tree. There are several ways to express the location of a set of nodes in a DOM tree using XPaths. As noted above, though the term XPath is used herein in a more precise and restricted sense. An XPath is considered herein as a path in a tree representation of a document that has the capability to include the positional information of the preceding siblings that have the same label as a given node in the tree.

The term *nodetest* is a label that defines a set of nodes (which are referred to as a node-set), in which each node in the set is a child node of the current node that has *nodetest* as its label. The label *table* is an example of *nodetest*. A *predicate* filters the node-set specified by the *nodetest* further into a smaller node-set and is always placed inside a pair of square brackets. [position()=index] and [position()<7] are examples of *predicates*.

Predicates of the form [position()=index] are abbreviated as [index]. A predicate is called an *equality predicate* if the predicate is of the form [position()=index]. Other predicates are termed *generalized predicates*. An XPath is called an *equality XPath* if all the terms in the XPath contain only equality predicates. If some of the terms in an XPath contain generalized predicates, the XPath is called a *generalized XPath*. For example, in **Fig. 3**, the XPath for node *3* is **/a[1]/b[1]/c[1]**. For node *6*, the XPath is **/a[1]/b[1]/d[1]/e[2]**.

Both of these XPaths are *equality XPaths*. The XPath of the form **/a[1]/b[1]/d[1]/e[position() < 3]**, is an example of a *generalized XPath* which evaluates to the node-set containing node 5 and node 6. Note that the last term of the XPath contains a *generalized predicate*. If any of the terms in an XPath contains a wildcard (that is, "*") or appears without a predicate, then the *Xpath* is still considered to be a *generalized XPath*. A *nodetest* of a *term* is referenced by *term.nodetest*. Similarly, the *predicate* can be referenced by *term.predicate*. An *index* of an *equality term* can be referenced by *term.index*.

An XPath not only captures the parent/child relationships between nodes but also incorporates some sibling information. For example, in an *equality XPath*, a positional index that occurs in a *term* indicates how many preceding siblings have the same nodetest (that is, have the same node label) in the DOM tree. An XPath, however, does not capture

sibling information of nodes whose node labels are different from that of the given node. For example, the XPath for node 6 in **Fig. 3** is **/a[1]/b[1]/d[1]/e[2]**. The positional index *2* for the node label *e* shows that this node has another sibling node with the label *e*. There is no reference hence to the sibling node *f*.

A document ***d*** can be defined by the set of all *equality XPaths* corresponding to the leaf nodes of the DOM tree for ***d***. Note that each *equality XPath* occurs only once in a document.

A dictionary $Dict_{XPath}$ can be constructed in a fashion similar to the one described in the *bag of paths* model, based on the *equality XPaths* for all the leaf nodes of the DOM trees in the document collection. Taking account of issues such as repetitions is not straightforward in this model. Elements that appear as a result of repetition differ in their positional indices, and thus have different XPaths. This is unlike the situation in the *bag of tree paths* model, in which all such paths are identical. Therefore, if the similarity measure defined for the *bag of tree paths* model is directly used in the *bag of XPaths* model, two documents that differ only in the number of repeating elements will have a relatively low similarity value.

If the number of repeating elements are different, then there are some XPaths that differ only in the positional index. An element gets its positional index based on how many preceding sibling nodes have the same label. For a case in which the number of repetitive elements is different in the two documents, there will be a different number of preceding siblings with the same label. Accordingly, an alternative similarity measure, which incorporates the issue of repetition for the *bag of XPaths* model, is now defined.

Similarity measure

A special type of defined *generalized predicate* is referred to as *repetitive predicate*. A *repetitive predicate* is of the form *[(position()-init) mod diff=0]*. [(position()-1) mod 5=0] is an example of *repetitive predicate*, in which the function *position()* returns the *positional index* of the node. The *index* values 1, 6, 11, etc. satisfy this *repetitive predicate*. In this example, the value of *diff* is 5 and the value for *init* is 1. Now, one may

observed that [(position()-1) mod 5 = 0] is satisfied for the 1st position, 6th position, and so on. That is, for positions that satisfy the expression [(6-1) mod 5] = 0].

A *generalized XPath,* which contains only *equality* and *repetitive predicates,* is called a ***repetitive XPath***. As such a repetitive XPath contains at least one *repetitive predicate* as defined above.

Table 1 below presents pseudocode that defines a Boolean function called *subsume*(X_1,X_2), in which X_1 and X_2 are XPaths (either *equality* or *repetitive*). The function returns "*true*" if the set of nodes evaluated by the XPath X_2 is a subset of the nodes evaluated by X_1 for the same tree. When the function returns "*true*", the XPath X_1 is said to subsume the XPath X_2.

As an example, consider two XPaths X_1=/tag1[1]/tag2[(position()-1 mod 5)=0] and X_2 =/tag1[1]/tag2[1]. All nodes evaluated by X_2 are also evaluated by X_1, and hence one concludes that the XPath X_1 subsumes the XPath X_2. The function *subsume* given in **Table 1** below does not evaluate the given XPaths on a tree, but uses another way to determine subsumption. In this algorithm, the function *depth* returns the number of *terms* present in the given XPath. The function *evaluate(p,i)* returns *"true"* if the *index i* satisfies the *predicate p*. Here, $term^j_i$ represents the j th *term* in the XPath X_i. The algorithm compares the given XPaths term by term and returns true if *predicates* for all the terms either match exactly or the *index* of second XPath is satisfied by the *predicate* of first XPath.

TABLE 1

boolean *subsume*(X_1 X_2) {
 if ($depth(X_1) \neq depth(X_2)$)
   return *false*
 flag = *true* ;
 for every term t_i of Xpath X_i do
  if ($term^t_1.nodetest \neq term^t_2.nodetest$)
   return *false*
  if ($term^t_1.predicate \neq term^t_2.predicate$)

```
        continue
      else
        let p be the term^t_1.predicate
        if (p is equality predicate)
          return false
        else
          flag = flag ∧ evaluate(p,term^t_2.index)
    return flag
  }
```

Table 2 below presents pseudocode that defines a function called *generalize(X_1,X_2)*. This code either returns a *generalized(repetitive)* XPath that subsumes both XPaths X_1 and X_2, or returns *"null"*. As an example, consider two XPaths X_1=**a[1]/b[1]/c[1]/d[1]** and X_2=**a[1]/b[1]/c[6]/d[1]**. The function *generalize(X_1,X_2)* returns the *generalized* XPath X_g = **a[1]/b[1]/c[(position() -1) mod 5 = 0]/d[1]**. The predicate c[(position() -1) mod 5 = 0] will evaluate to c[1], c[6], c[11] and so on.

TABLE 2

```
boolean generalize(X_1 X_2) {
  if (depth(X_1)  ≠ depth(X_2))
     return null
  gxpath = ""
  for every term t_i of Xpath X_i do
     if (term^t_1.nodetest  ≠ term^t_2.nodetest)
        return null
     if (term^t_1.index == term^t_2.index)
        gxpath = gxpath + "/" + term^t_1
        continue
     else
     diff = | term^t_1.index  ≠ term^t_2.index |
```

```
    init =term^t_1.index mod diff
    gxpath = gxpath + "/" +term^t_1nodetest + "[" + ((position() - init) mod diff) == 0 +
"]"
  return gxpath
}
```

A document d_i can be represented as a N bit binary vector where N is the number of terms in $Dict_{XPaths}$. Let D_{ei} denote the set of all *equality XPaths* that are present in document d_i.

Note that $D_{ei} \subset Dict_{XPaths}$. A set of generalized XPaths is generated, based on pairs of *equality XPaths* in D_{ei} using the algorithm defined in **Table 2**. Let D_{gi} denote the set of all *generalized XPaths* that are obtained using $generalize(X_1,X_2)$. The algorithm attempts to generalize two *equality XPaths* only if both of them have the same *tree path*, that is, the two *equality XPaths* without the *positional indices* must be identical. A *tree path* to XPath index is created so that for any given tree path, one can quickly obtain the set of *equality XPaths* that have the same *tree path.*

Let X denote one such set of *equality XPaths*, and let T be the number of *terms* in any $XPath \in X$. All the XPaths will have same number of terms. A pair of XPaths X_1 and X_2 are chosen in X, such that there exists some $i;\ 1 \leq i \leq T$ for which $term_1^i.index \neq term_2^i.index$. Here $term_1^i.index$ *and* $term_2^i.index$. are the lowest two indices for the $term_1^i.nodetest$ and $\forall k,\ k \neq i,\ term_1^k.index\ =\ term_2^k.index$.

That is, two XPaths are generalized if and only if they differ in the positional index at exactly one *term t*. Further, the two indices should be the lowest two indices that occur for the label associated with the term t in the set X. Therefore, the number of *generalized XPaths* that one can derive from a *tree path* is bounded by T.

The complete set of XPaths for document d_i is $D_{ei} \cup D_{gi}$. Now, the similarity measure for a pair of documents d_i and d_j can be defined as follows as expressed in **Equation [2]** below.

$$sim(d_i, d_j) = \frac{e+s}{n+m-e} \qquad [2]$$

In **Equation [2]** above, e is the number of XPaths that are common to both d_i, d_j. The term s is the total number of XPaths that do not exactly match but are subsumed by at least one of the generalized XPaths of the other document. The term n is the number of XPaths in d_i, and m is the number of XPaths in d_j. To compute s, the function *subsume* described above with reference to **Table 1** is used.

The *subsume* and *generalize* functions, given in **Tables 1** and **2** respectively, incorporate the issue of repetition in the *bag of XPaths* model. This model can accommodate other aspects, such as optional elements and recursive elements, by using other *subsume* and *generalize* functions. If the application at hand requires *optional* and *recursive* control structures to be incorporated in the similarity measure, suitable *subsume* and *generalize* functions for these control structures can be used. In other words, the functions *subsume* and *generalize* for specific control structures can be designed based on the application at hand.

Overview of procedure

Fig. 4 is a flowchart **400** that represents, in overview, steps in a simple example of comparing two documents. The flowchart **400** describes the procedure for obtaining the similarity between two documents d_i and d_l in a given document collection. These steps are as outlined below.

Step **410** Model all the documents as labeled trees and build a dictionary of paths or XPaths based on the *bag of Paths* or *bag of XPaths* model as required. Let $Dict\{p_1, p_2, \ldots, p_N\}$ be the dictionary of size N.

Step **420** Represent each document d_j in the collection as an N-dimensional vector $[d_{j1}, d_{j2}, \ldots, d_{jN}]$, where element i of the vector, that is, d_{ji} denotes the value of some feature associated with path p_i, such as the presence

or absence of path p_i, or the frequency of occurrence of path p_i in the document.

Step **430** Use the similarity measure given in **Equation [1]** or **Equation [2]** to obtain the similarity value between two documents based on the *bag of Paths* model or *bag of XPaths* model.

Implementation using computer hardware and software

Fig. 5 is a schematic representation of a computer system **500** that can be used to implement the techniques described herein. Computer software executes under a suitable operating system installed on the computer system **500** to assist in performing the described techniques. This computer software is programmed using any suitable computer programming language, and may be thought of as comprising various software code means for achieving particular steps.

The components of the computer system **500** include a computer **520**, a keyboard **510** and mouse **515**, and a video display **590**. The computer **520** includes a processor **540**, a memory **550**, input/output (I/O) interfaces **560**, **565**, a video interface **545**, and a storage device **555**.

The processor **540** is a central processing unit (CPU) that executes the operating system and the computer software executing under the operating system. The memory **550** includes random access memory (RAM) and read-only memory (ROM), and is used under direction of the processor **540**.

The video interface **545** is connected to video display **590** and provides video signals for display on the video display **590**. User input to operate the computer **520** is provided from the keyboard **510** and mouse **515**. The storage device **555** can include a disk drive or any other suitable storage medium.

Each of the components of the computer **520** is connected to an internal bus **530** that includes data, address, and control buses, to allow components of the computer **520** to communicate with each other via the bus **530**.

The computer system **500** can be connected to one or more other similar computers via a input/output (I/O) interface **565** using a communication channel **585** to a network, represented as the Internet **580**.

The computer software may be recorded on a portable storage medium, in which case, the computer software program is accessed by the computer system **500** from the storage device **555**. Alternatively, the computer software can be accessed directly from the Internet **580** by the computer **520**. In either case, a user can interact with the computer system **500** using the keyboard **510** and mouse **515** to operate the programmed computer software executing on the computer **520**.

Other configurations or types of computer systems can be equally well used to implement the described techniques. The computer system **500** described above is described only as an example of a particular type of system suitable for implementing the described techniques.

Example applications of document similarity measures

Application to information extraction

There has been much work in the area of information extraction from Web pages in the recent years. One approach to this problem is to generate a wrapper using an example page. The fields that contain the desired information are indicated by the user. A wrapper is created to capture the extraction rules based on the patterns exhibited by the indicated fields in the example page. The wrapper is then used to extract similar information from all the pages that are structurally similar to the given example page. This approach, however, requires that all structurally similar pages are first identified and grouped. The proposed similarity measure can be used to cluster pages based on structural similarity.

Application to Document Type Defining (DTD) induction

In the case of XML documents, knowledge of the DTD can facilitate the identification of structurally similar pages. Unfortunately, most of the XML documents on the Web are found without their DTDs. There are several induction algorithms that attempt to learn the DTD from a set of examples. These approaches assume that all the examples come from the same DTD. If the XML pages in a given collection come from different DTDs, these algorithms cannot be used directly, since it is theoretically infeasible to learn a single DTD for the entire collection. A possible solution is to partition the collection into smaller sets of "structurally similar" documents, and then learn the DTD for each set. Again, one can use the proposed similarity measure to cluster the pages based on "structural similarity".

Application to template removal

Common information contained in templates hinders the performance of many information retrieval and data mining algorithms. The common information is sometimes referred to as template information. Using the similarity measure described herein, one can determine an approach to identify this template information. The documents from a collection are first clustered based on their structure. A cluster contains pages that share a common look and feel. The text that appears at the same location in different pages within a cluster is identified as the common information.

Conclusion

A method, computer software, and a computer system are each described herein in the context of document structure comparison. Various alterations and modifications can be made to the techniques and arrangements described herein, as would be apparent to one skilled in the relevant art.